Parveen Khurana

124 Followers About Following

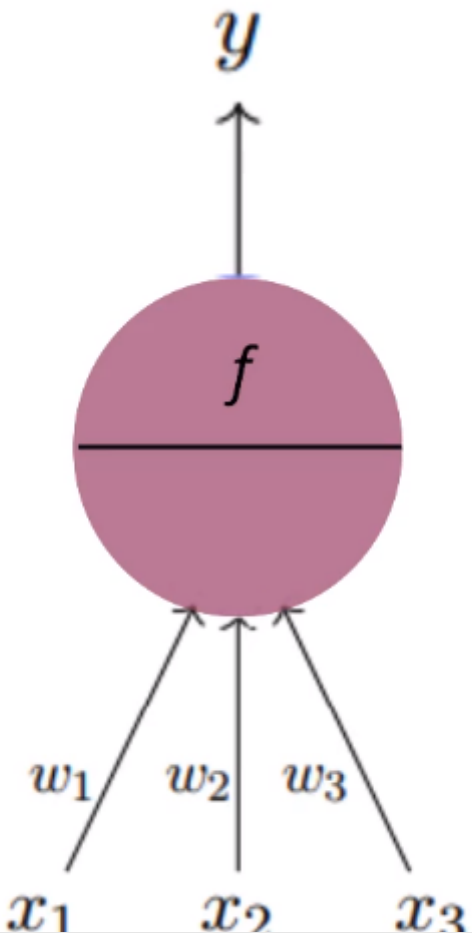
McCulloch Pitts(MP) Neuron

P Parveen Khurana Nov 30, 2019 · 13 min read

In this article, we discuss the 6 jars of the Machine Learning with respect to the MP Neuron model.

Introduction:

The fundamental block of Deep Learning is an Artificial Neuron.



Artificial Neuron

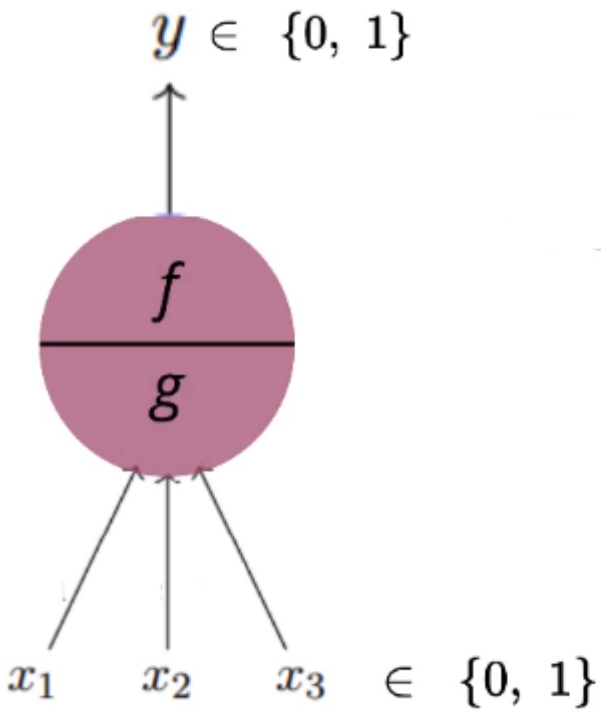
This is what an Artificial Neuron looks like. It takes on a bunch of **inputs**(say **x1, x2, x3 and so on**), these inputs would be factors/features based on which we make the decision and there are some weights assigned to each of the inputs. So, an **Artifical Neuron takes a weighted aggregate of the inputs(weights are just 1 in case of MP Neuron) and it applies some function on this weighted aggregate input and gives an output.**

The MP Neuron model is also known as a linear threshold gate.

MP Neuron Model

A model is our approximation of the true relationship that exists between **x** and **y**. Speaking of a model, we must know what the function is, what the parameter values are(that we will eventually learn from data), what kind of inputs this function takes and what kind of output it gives.

For MP Neuron Model, **inputs can only be boolean** that means belongs to the set **(0, 1)**. Similarly, the model is going to **output Boolean value**. In other words, neither the inputs nor the output can be some other real number.



g sums up all the inputs(weighted sum) and then f takes g as the input.

In this case, as we just sum up all the inputs for **g** and since all the inputs are Boolean, which means we are basically counting the number of things which are **on**(**have a value of 1**) in the input set; that's what the summation means in this case when all the weights are just 1.

Now, this value of **g** we pass it to other function **f** which would **output 1(means neuron would fire) if the summation of inputs(which is stored in g) is greater than some threshold and it will output 0 if the summation of the inputs is less than some threshold.**

There is also something known as **inhibitory input**, **if that is on, then no matter what the other inputs are, the neuron is not going to fire**. For example, let's say the decision to make is whether to go out or not. So, this decision depends on several factors like weather, free time and so on but if let's say that person has a high fever on that day then no matter what other conditions are, we can make the decision as not going out today. **So, inhibitory input is something that overrides all other inputs.**

The **only parameter of this model** is **b(threshold value)**.

- McCulloch and Pitts proposed a highly simplified computational model of the neuron.
- ***g*** aggregates the inputs and the function ***f*** takes a decision based on this aggregation.
- The inputs can be excitatory or inhibitory

$y = 0$ if any x_i is inhibitory, else

$$g(x_1, x_2, ...x_n) = g(x) = \sum_{i=1}^{n} x_i$$

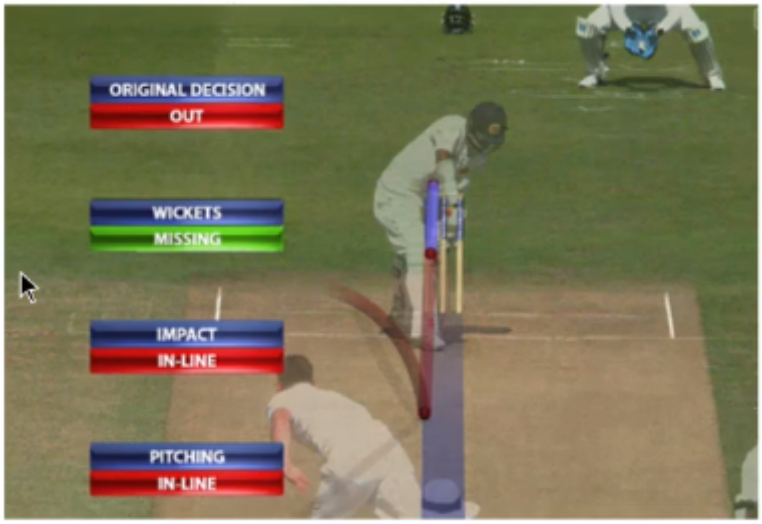
$$y = f(g(x)) = 1 \text{ if } g(x) \geq b$$

$$= 0 \text{ if } g(x) < b$$

One parameter, b

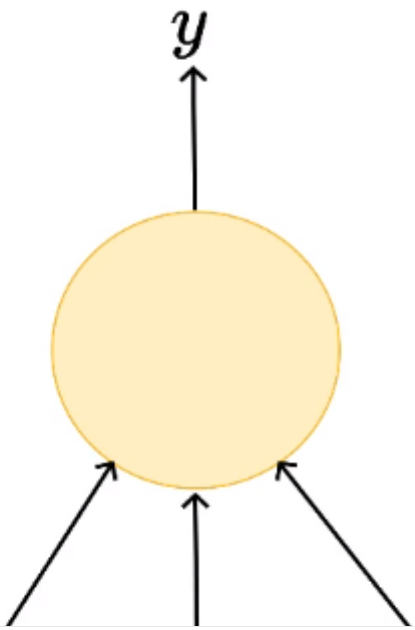
MP Neuron Data Task

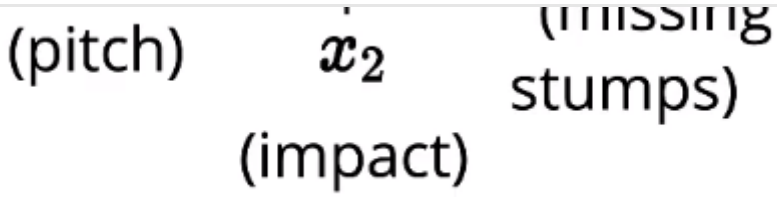
Since the data for the MP Neuron model is Boolean, there are some restrictions(we have to see those tasks where the data would be boolean) in some sense regarding the



In this case, the data could be represented in the Boolean format as below:

Pitch in line	Impact	Missing stumps	Is it LBW? (y)
1	0	0	0
0	1	1	0
1	1	1	1





There might be other factors affecting LBW decision but here we are considering only these three features in this case.

$$y = \left(\sum_{i=1}^{3} x_i \geq b\right)$$

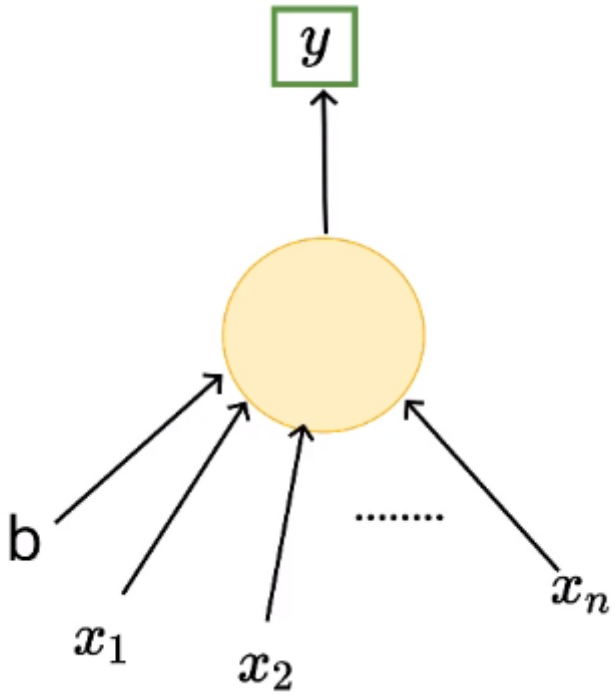
We need to find the value of **b** in such a way that when all the **x values** are plugged in the equation along with the **b** value, the predicted value of **y** must match the true output.

Let's take another example where the data is not naturally boolean:

There could be one feature which tells when the phone was launched and it could be like it was launched 15 days back or 1 month back and so on. Now these are not Boolean inputs for sure, and this input really matters in decision making whether we should buy this phone or not, so even in this type of cases, we could convert data in a form which looks boolean, we could convert this into a feature which tells us whether it was launched within the last 6 months.

Launch (within 6 months)	0	1	1	0	0	1	0	1	1
Weight (<160g)	1	0	1	0	0	0	1	0	0
Screen size (<5.9 in)	1	0	1	0	1	0	1	0	1
dual sim	1	1	0	0	0	1	0	1	0
Internal memory (>= 64 GB, 4GB RAM)	1	1	1	1	1	1	1	1	1

Battery(>3500mAh)	0	0	0	1	0	1	0	1	0
Price > 20k	0	1	1	0	0	0	1	1	1
Like (*y*)	**1**	**0**	**1**	**0**	**1**	**1**	**0**	**1**	**0**



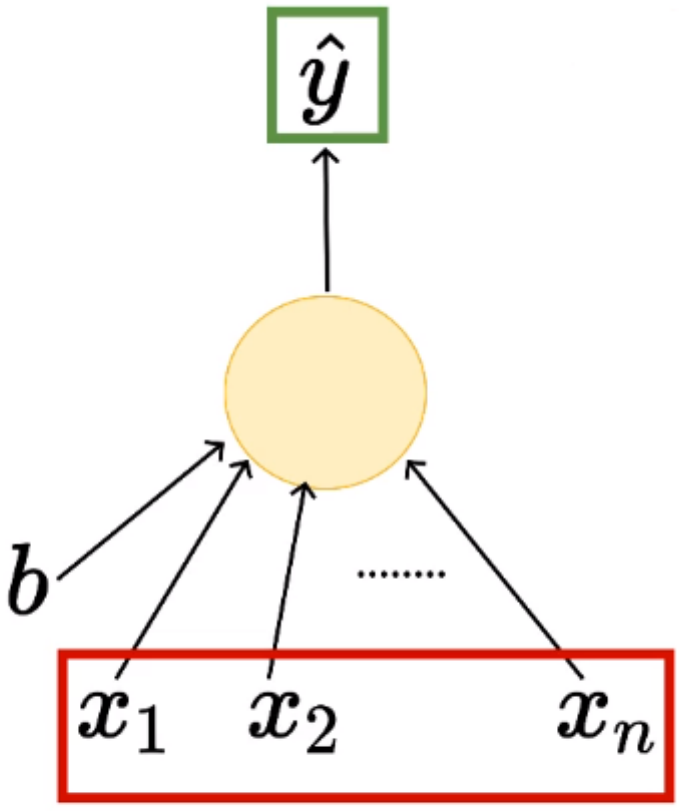
Here b is an input to the decision-making process, that's what we mean by indicating b here

$$y = \left(\sum_{i=1}^{n} x_i \geq b\right)$$

- Boolean inputs
- Boolean output

MP Neuron Loss

									
Launch (within 6 months)	0	1	1	0	0	1	0	1	1
Weight (<160g)	1	0	1	0	0	0	1	0	0
Screen size (<5.9 in)	1	0	1	0	1	0	1	0	1
dual sim	1	1	0	0	0	1	0	1	0
Internal memory (>= 64 GB, 4GB RAM)	1	1	1	1	1	1	1	1	1
NFC	0	1	1	0	1	0	1	1	1
Radio	1	0	0	1	1	1	0	0	0
Battery(>3500mAh)	0	0	0	1	0	1	0	1	0
Price > 20k	0	1	1	0	0	0	1	1	1
Like (y)	1	0	1	0	1	1	0	1	0
Prediction $\hat{y}$	0								



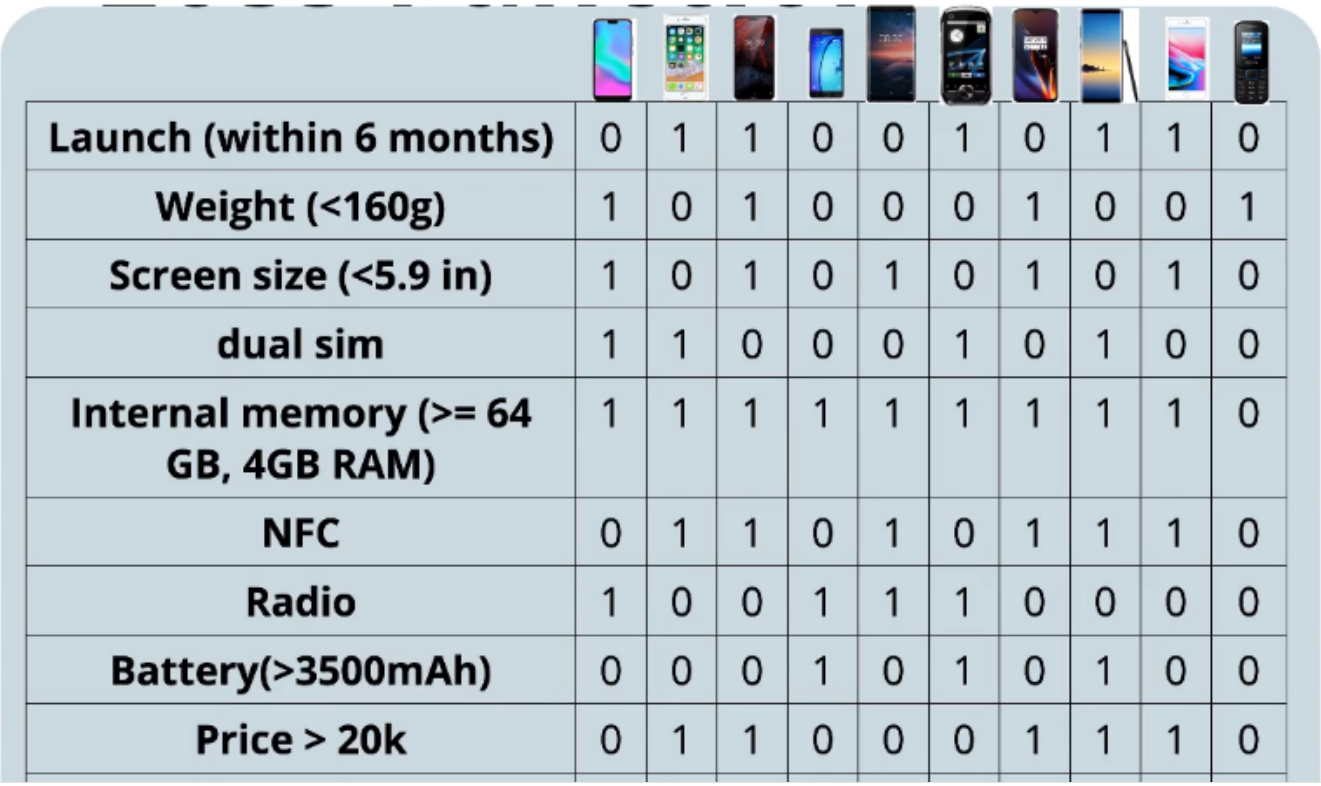
Now for this first data point, the predicted output is different from the true output. So, we could say that the error, in this case, is the difference between the true value and the predicted value.

$$loss/error = y - \hat{y}$$

So, this is one way of computing the error. And we have computed this for the first data point.

We can have the summation of this loss over all the data points.

$$loss = \sum_i y_i - \hat{y}_i$$



Launch (within 6 months)	0	1	1	0	0	1	0	1	1	0
Weight (<160g)	1	0	1	0	0	0	1	0	0	1
Screen size (<5.9 in)	1	0	1	0	1	0	1	0	1	0
dual sim	1	1	0	0	0	1	0	1	0	0
Internal memory (>= 64 GB, 4GB RAM)	1	1	1	1	1	1	1	1	1	0
NFC	0	1	1	0	1	0	1	1	1	0
Radio	1	0	0	1	1	1	0	0	0	0
Battery(>3500mAh)	0	0	0	1	0	1	0	1	0	0
Price > 20k	0	1	1	0	0	0	1	1	1	0

loss	0	0	1	-1	0	0	-1	1	0	0

Now we have computed the predicted values for all the data points and based on that we can compute the loss for each of the data points. But if we add up the loss for all of the data points, the net value would be 0 for the above case as the -ve values would cancel out the +ve values. So, it is never a good idea to consider the value of individual loss.

So, instead of taking the true difference, we take up the square of the difference(we could also take the modulus of the difference but the problem with the modulus function is that it is not differentiable).

$$loss = \sum_i (y_i - \hat{y}_i)^2$$

So, considering the squared error loss, the loss value would come out to be 4 in the above case.

MP Neuron Learning Algorithm:

We want to find the optimum value of **b** in such a way that when this value is plugged into the model, then the overall loss is minimized.

In this case, as we have only one parameter, we can use the Brute Force Search.

Let's say we have a total of **n** features on which we are basing our decision, so that means the summation of all the values(value of function **g**) can only take on values between **0 and n**. Moreover, the sum would only be distinct values, it would not be like 0.1 or 0.2 and would be an integer between 0(it would be 0 when all the features are off) and n(it would be n when all the features are on). So, this means **b would lie between 0 and n**.

As we are going with the Brute Force search, let's say **we start with the value of b as 0**, then in that case, for the first data point, we have the summation of the features as below:

Weight (<160g)	1	0	1	0	0	0	1	0	0	1
Screen size (<5.9 in)	1	0	1	0	1	0	1	0	1	0
dual sim	1	1	0	0	0	1	0	1	0	0
Internal memory (>= 64 GB, 4GB RAM)	1	1	1	1	1	1	1	1	1	0
NFC	0	1	1	0	1	0	1	1	1	0
Radio	1	0	0	1	1	1	0	0	0	0
Battery(>3500mAh)	0	0	0	1	0	1	0	1	0	0
Price > 20k	0	1	1	0	0	0	1	1	1	0
Like? (y)	1	1	1	0	0	1	1	1	0	0
prediction	?	?	?	?	?	?	?	?	?	?

Summation of all features for the first data point would be 5, we check $5 \geq 0$, and since it is true, we predict the output for this data point as 1.

We can continue this procedure for **all the 'm' inputs(data points)** that we have and compute the prediction for all the data points.

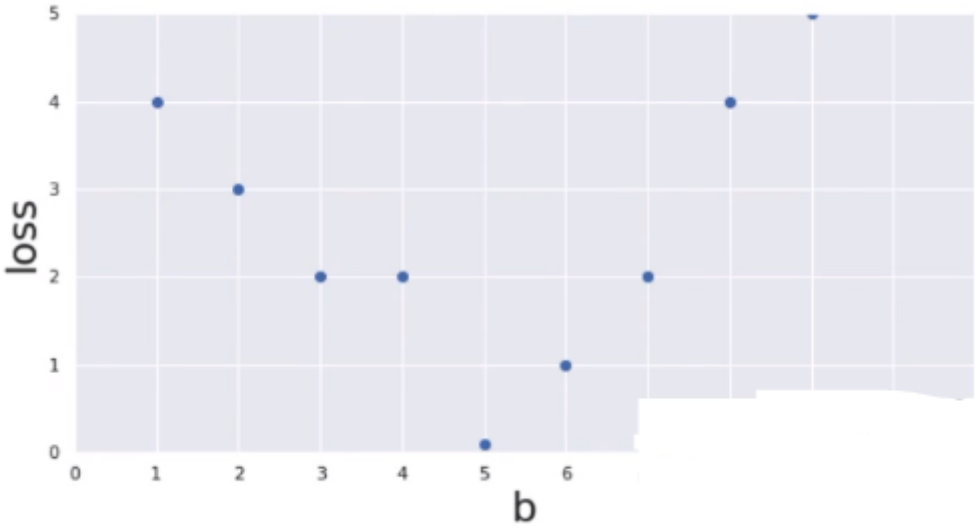
Then **we take the value of b as 1** and we compute the output for each of the data points and using that we can easily compute the loss and for the below case the answer is going to be 4 as the loss value.

Launch (within 6 months)	0	1	1	0	0	1	0	1	1	0
Weight (<160g)	1	0	1	0	0	0	1	0	0	1
Screen size (<5.9 in)	1	0	1	0	1	0	1	0	1	0
dual sim	1	1	0	0	0	1	0	1	0	0
Internal memory (>= 64 GB, 4GB RAM)	1	1	1	1	1	1	1	1	1	0
NFC	0	1	1	0	1	0	1	1	1	0
Radio	1	0	0	1	1	1	0	0	0	0
Battery(>3500mAh)	0	0	0	1	0	1	0	1	0	0
Price > 20k	0	1	1	0	0	0	1	1	1	0

prediction	1	1	1	1	1	1	1	1	1	1

Similarly, we compute the loss for the threshold value(b value) as 2, 3 and so on all the way up to n.

Then we can plot the threshold value against the Loss value.



And based on the plot we can get the optimum value of the threshold.

MP Neuron Evaluation:

We will use the threshold value as 5 and pass in the test data to the model.

Training data

Launch (within 6 months)	0	1	1	0	0	1	0	1	1	0
Weight (<160g)	1	0	1	0	0	0	1	0	0	1
Screen size (<5.9 in)	1	0	1	0	1	0	1	0	1	0
dual sim	1	1	0	0	0	1	0	1	0	0
Internal memory (>= 64 GB, 4GB RAM)	1	1	1	1	1	1	1	1	1	0
NFC	0	1	1	0	1	0	1	1	1	0
Radio	1	0	0	1	1	1	0	0	0	0
Battery(>3500mAh)	0	0	0	1	0	1	0	1	0	0
Price > 20k	0	1	1	0	0	0	1	1	1	0
Like? (y)	1	1	1	0	0	1	1	1	0	0
predicted	1	1	0	1	1	1	1	0	0	0

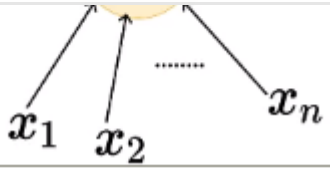
Test data

1	0	0	1
0	1	1	1
0	1	1	1
0	1	0	0
1	0	0	0
0	0	1	0
1	1	1	0
1	1	1	0
0	0	1	0
0	1	0	0
0	1	1	0

$$\hat{y} = 1 \text{ if } \sum_{i=1}^{n} x_i \geq b$$

$\hat{y}$

$$loss = \sum_i (y_i - \hat{y}_i)^2$$



$$Accuracy = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}}$$

Test data

1	0	0	1
0	1	1	1
0	1	1	1
0	1	0	0
1	0	0	0
0	0	1	0
1	1	1	0
1	1	1	0
0	0	1	0
0	1	0	0
0	1	1	0

Test data

1	0	0	1
0	1	1	1
0	1	1	1
0	1	0	0
1	0	0	0
0	0	1	0
1	1	1	0
1	1	1	0

0	1	1	0

Test data

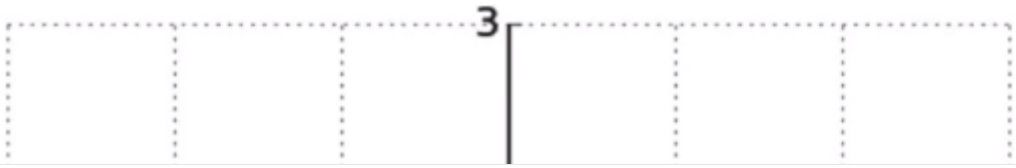
1	0	0	1
0	1	1	1
0	1	1	1
0	1	0	0
1	0	0	0
0	0	1	0
1	1	1	0
1	1	1	0
0	0	1	0
0	1	0	0
0	1	1	0

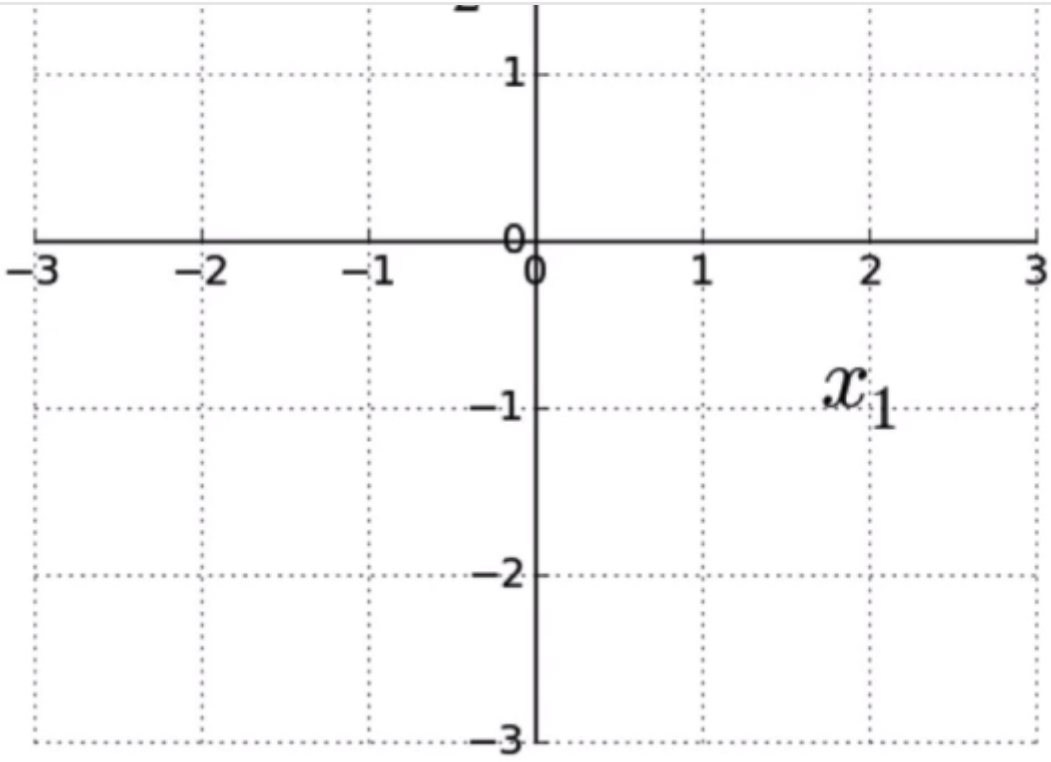
$$Accuracy = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}}$$

$$= \frac{3}{4} = 75\%$$

MP Neuron Geometric Interpretation:

Let's say we have two input features x1 and x2. We can plot this on a co-ordinate axis as below:





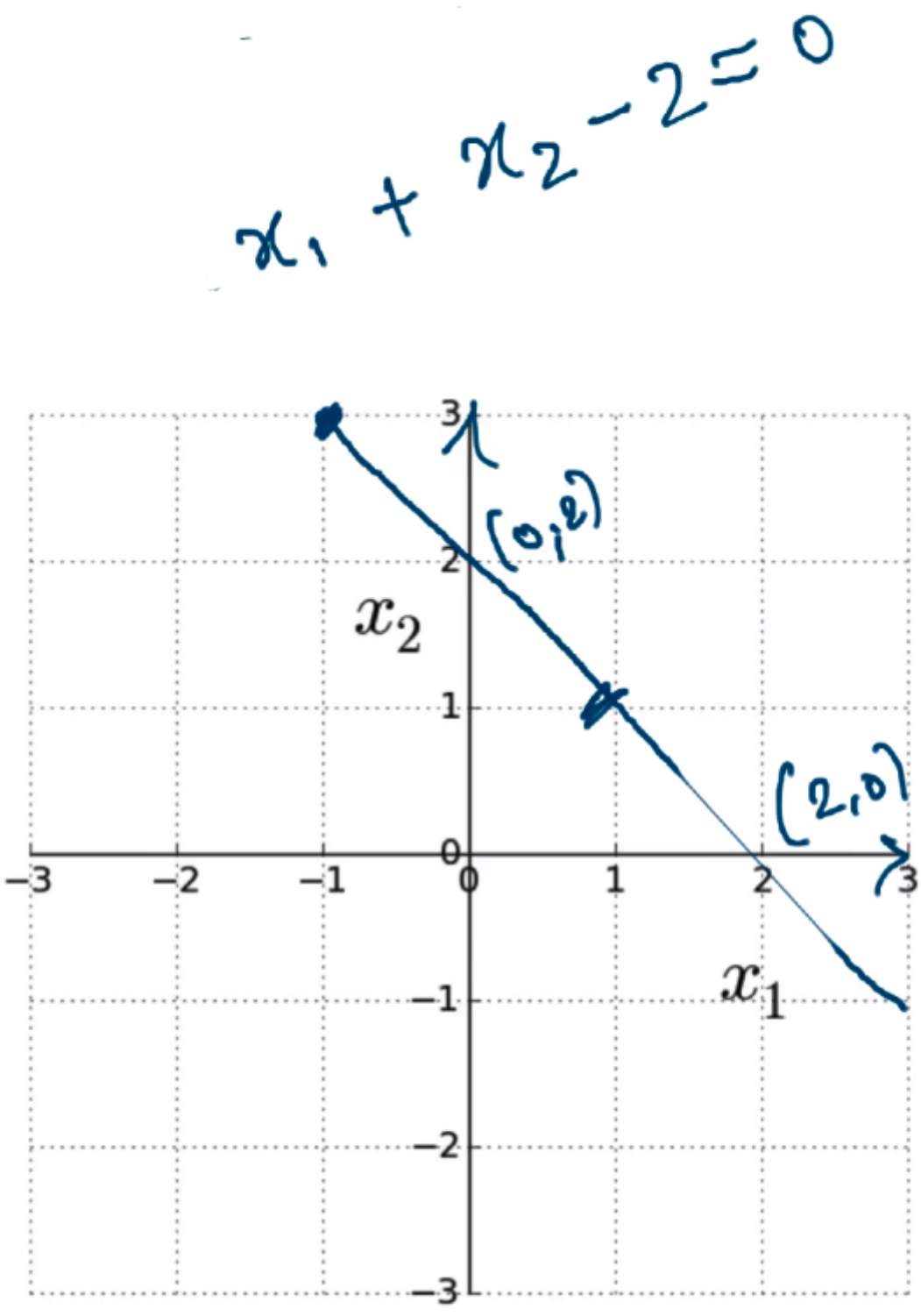
Generally, we have the x-y plots, but in ML, **y** is generally used to denote the output so we can represent two axes as x1 axis and x2 axis where x1 and x2 are the input to the model.

Now, in this case, the equation of the line would be:

$$x_2 = m x_1 + c$$

And we can look at this equation like

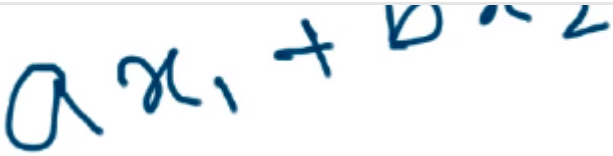
$$m x_1 - x_2 + c = 0$$



All the points on this line would satisfy the equation of the line for example: (-1, 3) lies on the line and we have

- -1+3-2 = 0 (x1 + x2–2 = 0)

The general form of a line we can write as:



For the above case, a = 1, b = 1 and c = -2.

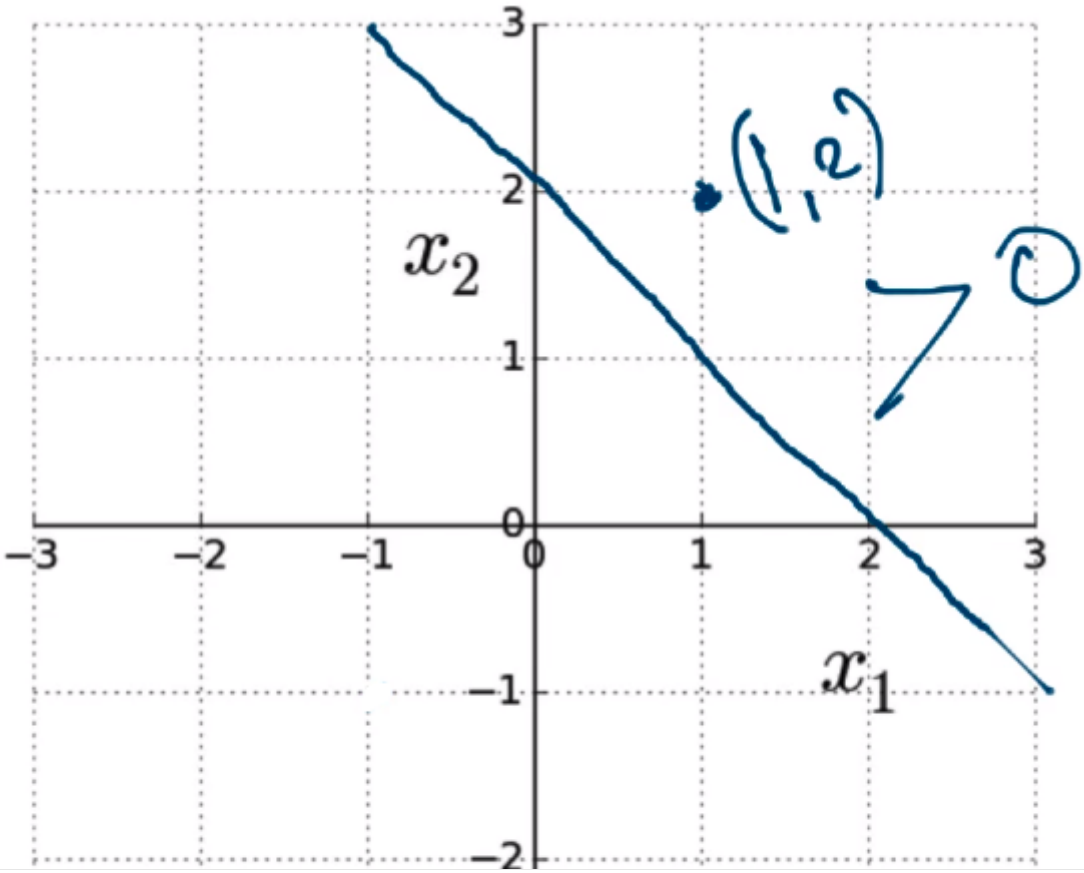
Now let's look at some points which are above the line and which are below the line and plug those points in the equation of the line.

Let's take point (1, 2) which lies above the line, plugging this into the equation of the line, we have:

1+2–2 = 1 > 0

So, **for any point above the line, the answer to the line equation(after plugging in the point in the equation of the line) is positive. That means all points lying above the line satisfies the equation:**

ax + by + c > 0

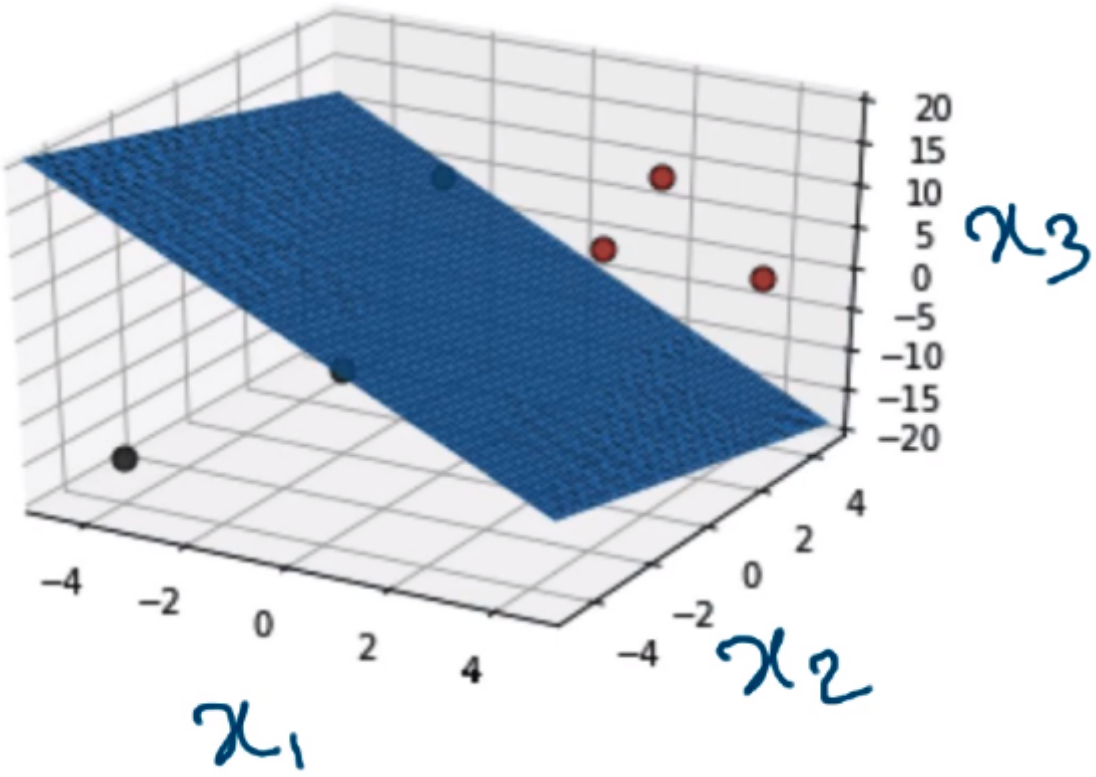


Similarly, **all the point lying below the line satisfies the equation:**

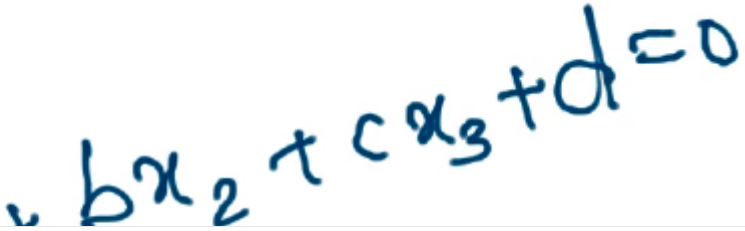
ax + by + c < 0

This was all in the 2D, let's look at the case of **3D** where we have **3 inputs: x1, x2, and x3**. And now we think of the equation of the line/plane as

ax1 + bx2 + cx3 + d = 0



Let's say this blue line/plane represents our 3D equation.



So, again the same condition holds, **all the points on the plane/line satisfies the condition of plane/line, all the points lying above the plane/line satisfies the condition ax1 + bx2 + cx3 + d > 0 and all the points lying below the line satisfies the condition ax1 + bx2 + cx3 + d < 0.**

So, in effect, what a line does in 2 dimensions is that it separates all the points on one side from the points on the other side. It acts as a boundary. Similarly, a plane in 3 dimensions separates all the points in that 3-dimensional space which lies on one side of the plane and which lies on the other side of the plane. It acts as a boundary. And the same analogy holds for higher dimensions also where if have an **n**-dimensional plane, it will separate all the points in that dimensional space into two sides.

So, let's see how all this relates to MP Neuron.

$$\hat{y} = (\sum_{i=1}^{n} x_i \geq b)$$

The above is the function of the MP Neuron, if this sum is greater than or equal to '**b**' then the output would be 1 otherwise it would be 0.

And the above function in 2 dimensional plane would be **x1 + x2 — b ≥ 0**.

Let's consider the equality part first: **x1 + x2 — b = 0**

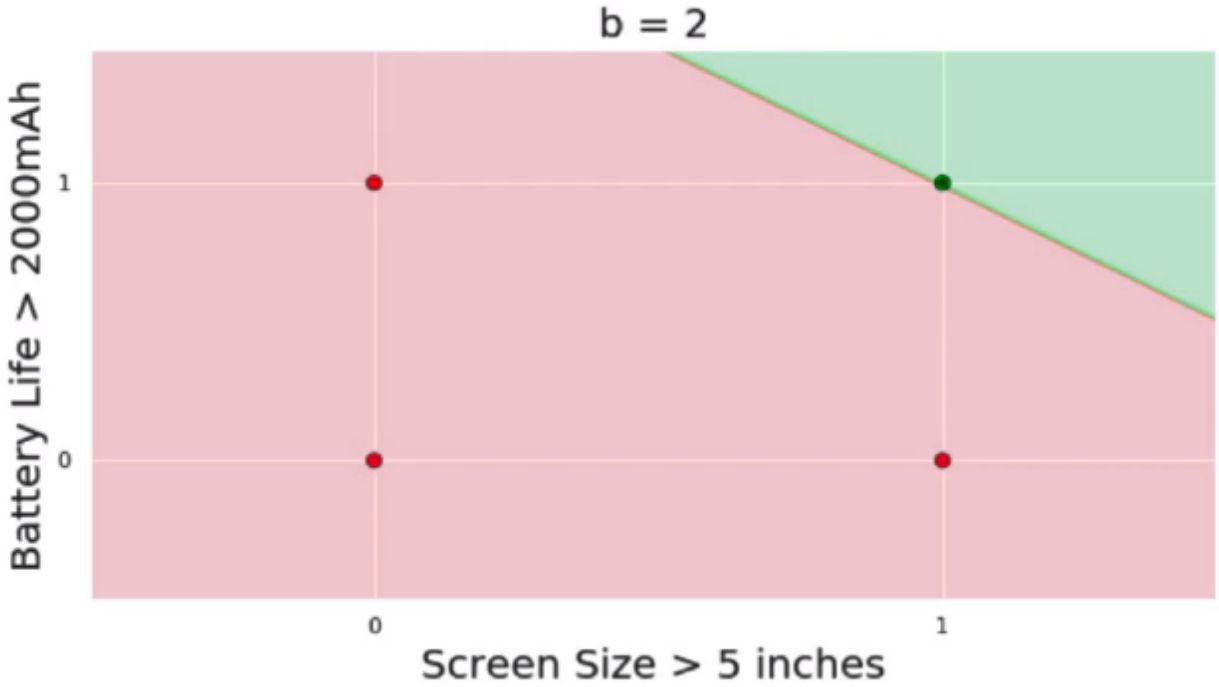
All the points which lie on this line would satisfy the above equality and all the points which lie above this line would satisfy the below equation:

x1 + x2 — b > 0

And all the points which lie below this line would satisfy **x1 + x2 — b < 0**

And that's exactly what MP Neuron does. It divides the points into two halves, one set of points which lie above the line and the other set of points which lie

Screen size (>5 in)	1	0	1	0	1	0	1	0	1	0
Battery (>2000 mAh)	0	0	0	1	0	1	0	1	0	0
Like	1	0	1	0	1	1	0	1	0	0

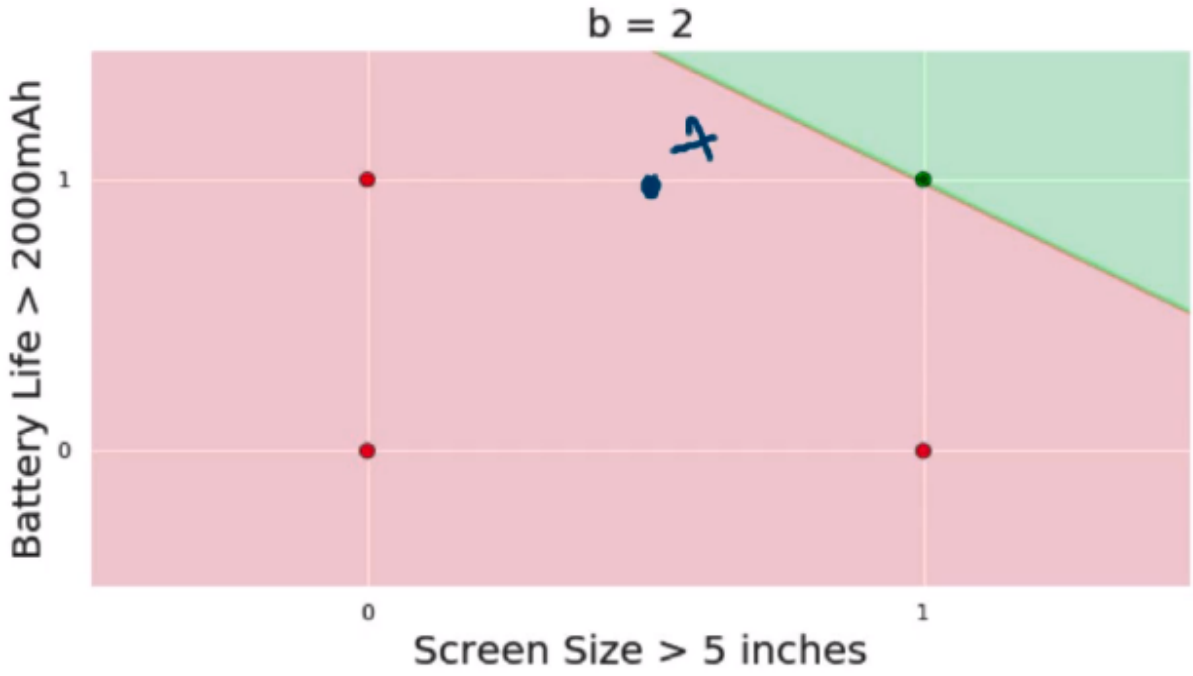


In the above case, we have two features: Battery Life and Screen Size and since we can only have Boolean inputs, there are only 4 combinations possible: either both the features 0 value i.e (0, 0) or we have (0, 1) or (1, 0) or (1, 1).

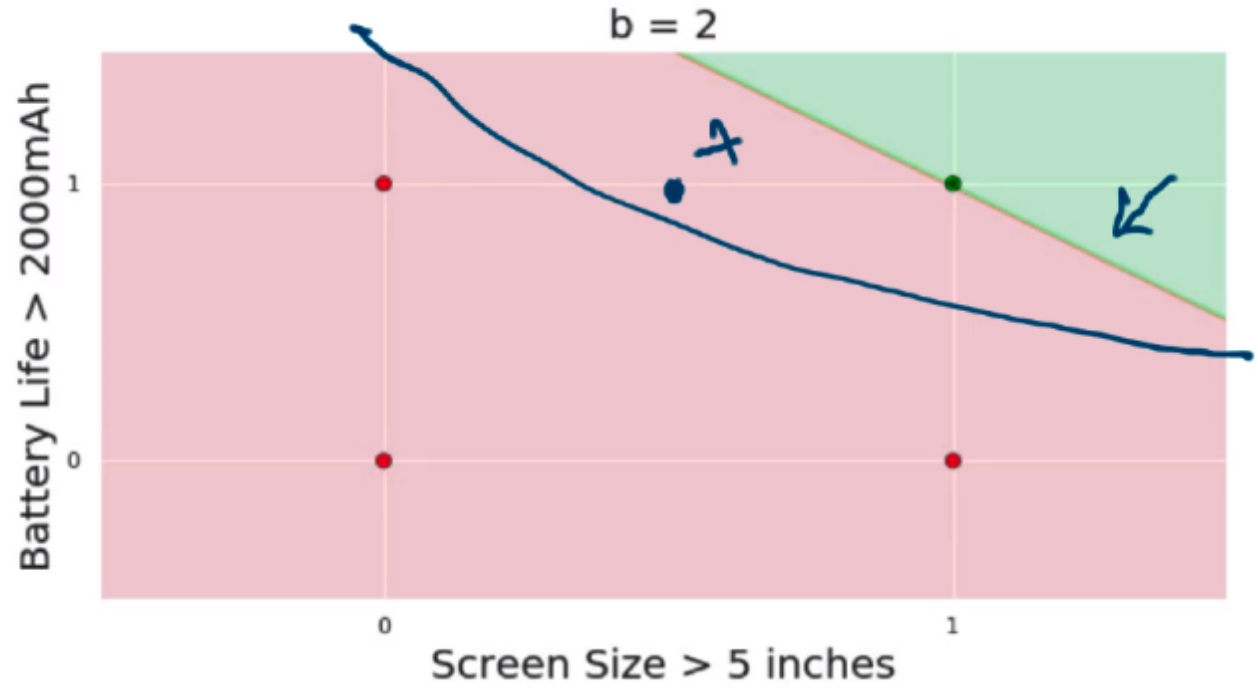
So, even if we have 10 phones(data points) in the above image, if we plot them, then all of that would lie on any of the 4 data points((0, 0), (0,1), (1, 0), (1, 1)) mentioned above.

And now if we train our model on this, what the model does is that it tries to find the equation of the line such that all the positive data points(with true output as 1) lie on this line or above that, that means all the positive points satisfy the equation $x1 + x2 - b \geq 0$. And the points(for which output was -ve) lies below the line.

say we have a point(in blue) as in the below image which was positive(true output 1)



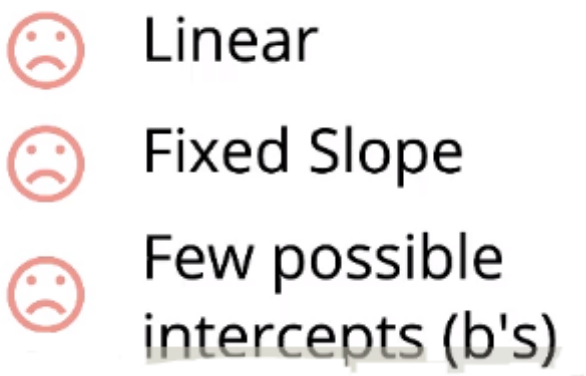
In this case, we would like our line to have a different slope instead of -1(below image) so that this point is also classified as +ve, this flexibility is not there.



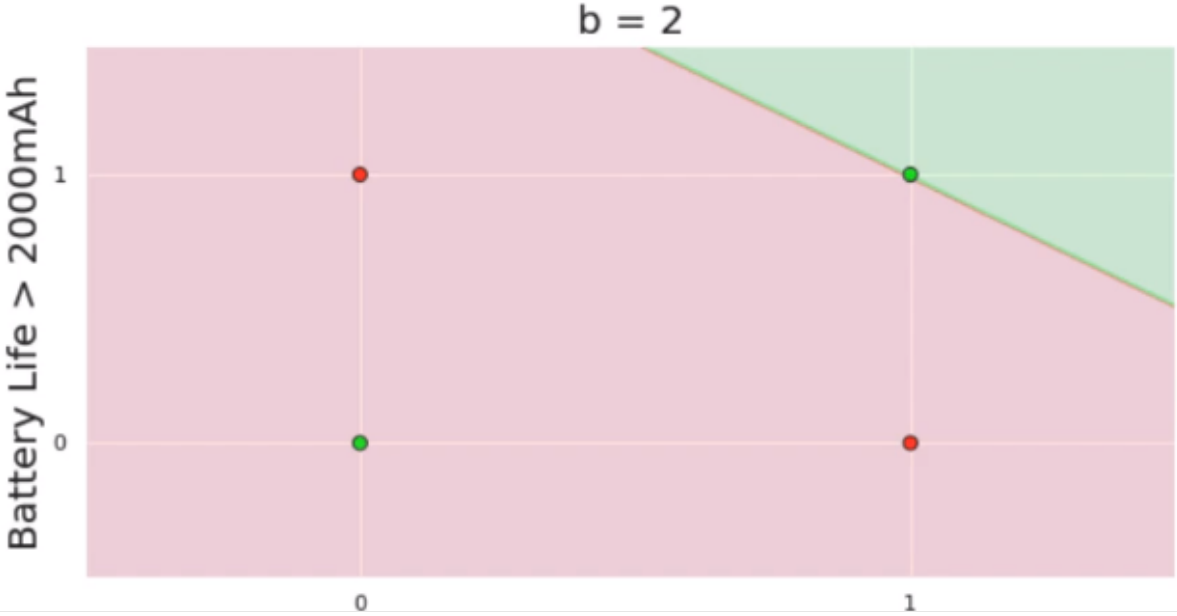
And the reason we want this line to have a different slope is that these two positive points lie on one side and the three negative points lie on the other side.

restrictive model, it does not give us any flexibility with the slope of the line and y-intercept can also pass through a fixed set of values in this case, b could be 2, 1, or 0. All the intermediate values are not possible. **So, this is a very restrictive model in terms of the freedom it has with respect to its parameters. The only parameter we can change is b and that too any discrete value from (0, 1, 2) or in general any value in the set 0, 1, …, up to n where n is the number of features that we have.**

In summary, we have that this model is actually linear, it finds a linear boundary between the positive and negative points, it has a fixed slope and it has very few possible intercepts(b values).

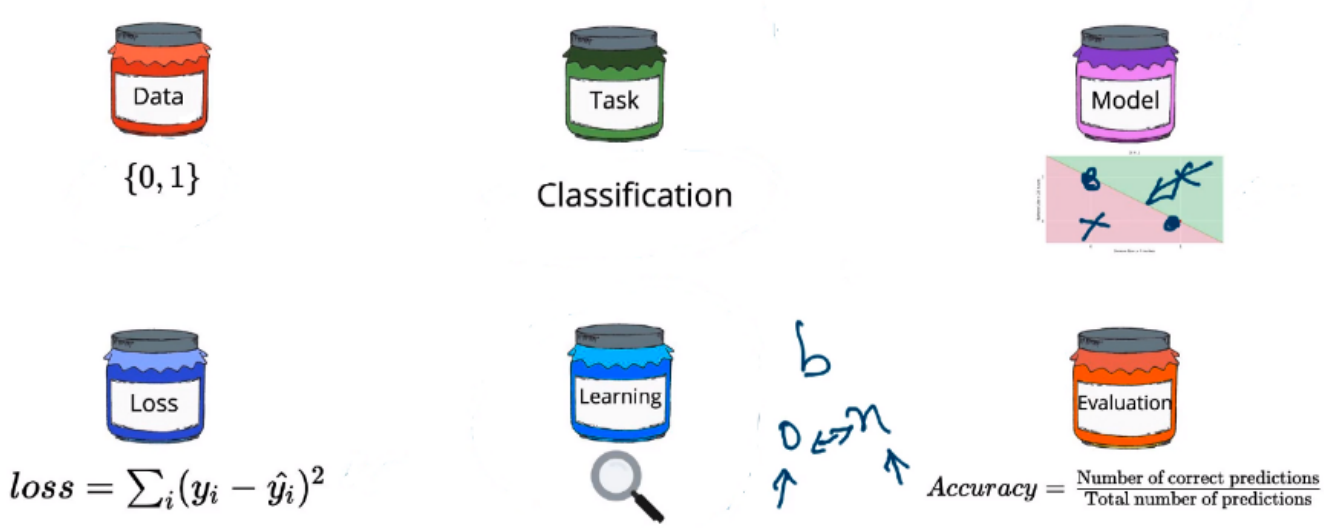


Now one immediate limitation is that if we take this new data(below image), let's say that we are okay with phones which has a small screen size and a small battery life or if they have large screen size and large battery life:



We cannot draw a line that separates the positive and negative points in this case. So, even with this simplistic example where we have only two features, there is this scenario, where we have certain points as +ve and certain points as -ve, and we can not find an MP Neuron(or to say value of b) which can separate the data points and give us 0 error on training data.

So, all the six jars of MP Neuron looks like this:



Data that we can deal with is **Boolean inputs**.

The **task** that we can do it for is **Binary Classification**(the output could be either 0 or 1).

The **model** tries to find the equation of a line such that all the positive points(true output as 1) lies on or above the line and the negative points(true output as 0) lies below the line.

The **loss function** was simple and the key take away here was that we should not use the exact difference, we should either take the absolute difference or the square difference. And the reason we don't use the absolute difference is that because that function is not differentiable.

Learning: We search for the optimum value of b, we go through all the values that the parameter b can take(it could take value from 0 to n where n is the number of features) and see the loss that we get for each of the values of b.

predictions.

To overcome some of the limitations of the MP Neuron model, we look at the Perceptron model.

References: PadhAI

Machine Learning Deep Learning Artificial Intelligence Artificial Neuron

Mcculloch Pitts Neuron